ADAPTIVE PRODUCTION OF ASSEMBLER

Field of the invention

The present invention relates to a system for generating an assembler for a target microprocessor, a method of assembling a machine language program, an assembler for a microprocessor and to a method of preparing a program for execution on a microprocessor.

Background of the invention

During the preparation of a binary machine language program for execution on a microprocessor a number of operations are effected. Such operations include translation of a representation of the program in a source language into the binary machine language, binding symbols to addresses, and debugging the program. The process of translation may be accomplished by a compiler which receives as its inputs a high level language representation of the program, or an assembler which receives as its inputs an assembly language representation of the program.

A problem exists in the preparation of programs for microprocessors which are still in the process of development in that the architecture of the instruction set may alter during development of the microprocessor. Such changes may take the form of altering the size and location of instruction operands or the writing of new instructions.

Assemblers are typically programs which translate instructions comprising mnemonics and operands into binary representations and which translate the mnemonics and operands (including immediate data) into corresponding binary values, viz opcodes and encoded operands. Each instruction comprises a set of contiguous bit fields which fully characterize the instruction, where a bit field is a

sequence of contiguous bits.

The assembler must ensure for example that the binary representation of operands are located in the correct bit field and this is typically achieved by hard coding of control information into the assembler program. Likewise, if encoding a particular operand involves a specified operation to encode it, for example, division by 2, this operation is typically hard coded into the assembler.

A problem which arises if the instruction set architecture is changed is that re-writing of the features hard coded into the assembler, amongst other things, will be necessary to account for the changes.

It is an object of the present invention to at least partially overcome the difficulties of the prior art.

It is an object of some embodiments of the invention to provide a system for generating an assembler which system takes into account updates in processor instruction architecture so that the resultant assembler complies with such updates.

Summary of the invention

According to one aspect of the invention there is provided a system for generating an assembler for a target microprocessor, the system comprising:-

a descriptor file containing information descriptive of the instruction set of said target microprocessor and a translation device for translating assembly language into machine language as an output wherein the translation device comprises a fetching device for acquiring data from said descriptor file and a control device receiving said data from said fetching device and constraining the output

of said translation device to conform to the architecture of said instruction set.

Preferably the descriptor file further comprises syntax information for each instruction, and the translation device translates each instruction on the basis of said syntax information.

Advantageously the system further comprises a data capture device having an input for accessing the instruction set of said target microprocessor and having an output, wherein said output comprises said descriptor file.

Preferably the system further comprises a linker, wherein the system has a data transfer device outputting selected data fetched from said descriptor file to said linker, whereby said linker uses said output data to modify the translated output of said system.

According to a second aspect, the invention comprises a method of assembling a machine language program for a target microprocessor comprising:-

providing a descriptor file containing information descriptive of the instruction set of said target microprocessor;

translating assembly language instructions into machine language wherein the translation step comprises

acquiring data from said descriptor file; and

constraining the machine language to conform to the architecture of said instruction set.

Advantageously said descriptor file further contains syntax

information for each possible instruction of the instruction set, and said translating step comprises transliterating each assembly language instruction using said syntax information.

According to another aspect the invention provides a method of preparing a program executable on a target microprocessor comprising:

capturing data from the instruction set of said target microprocessor thereby forming a descriptor file containing information descriptive of said instruction set;

providing assembly language instructions for said target microprocessor;

translating each assembly language instruction into a corresponding machine language output; and

using data from said descriptor file, constraining the machine language output to conform to the architecture of said instruction set.

According to a further aspect, the invention provides a method of preparing a program executable on a microprocessor, comprising:

providing plural program modules, at least one of said modules having one or more instructions including external symbols, wherein external symbols have values which cannot be determined without reference to another program module;

providing a descriptor file containing information descriptive of the instruction set of said target microprocessor;

translating assembly language instructions into machine language wherein the translation step comprises

acquiring data from said descriptor file;

constraining the machine language to conform to the architecture of said instruction set;

and further comprising binding external symbols to addresses using data selected from said descriptor file.

providing a descriptor file containing information descriptive of the instruction set of said target microprocessor;

translating assembly language instructions into machine language wherein the translation step comprises

acquiring data from said descriptor file;

constraining the machine language to conform to the architecture of said instruction set;

and further comprising binding external symbols to addresses using data selected from said descriptor file.

Brief description of the drawings

A preferred embodiment of the present invention will now be described, by way of example only, with reference to the accompanying drawings in which:-

Figure 1 shows a block diagram indicating the context of the present invention.

Figure 2 is a block diagram showing an exemplary preferred

embodiment to the invention.

Figure 3 shows an exemplary instruction from the instruction set of Figure 2.

Figure 4 shows an entry in the description file shown in Figure 2 corresponding to the instruction of Figure 3.

Figure 5 shows a second exemplary instruction of the instruction set of Figure 2 and;

Figure 6 shows a second entry in the description file of Figure 2 corresponding to the instruction of Figure 5.

Description of the preferred embodiment

Every type of microprocessor has its own machine language which consists entirely of numbers and is very difficult to directly read or write. For this reason, it is normal when writing programs to write program modules in a high level language which is to a greater or lesser extent independent of the microprocessor. Then the program or program module is translated into machine language using a translation device known as a compiler.

In certain situations however it is necessary to write programs which are directly analogous to the machine language by writing detailed instructions in assembly language. Assembly language is, like machine language unique to each type of microprocessor but instead of being written in numbers, comprises mnemonic commands each corresponding to one of the microprocessor opcodes, together with operands. Operands can either be numbers or names used to make a symbolic reference to a number, typically the address of some named location in memory.

After writing a program module in assembly language, the resultant text file is translated using an assembler into machine language.

Instructions consist of a number of bit fields each representing different information required to carry out an operation. Such fields include opcodes, operands and fields reserved for architecture use, the operands including register designators and immediate data. For any one microprocessor it is possible for instructions to have different formats appropriate to the operation being performed. Thus, one opcode may require two operands whereas another opcode may merely require a single operand. Furthermore the size of a bit field available for the operand is likely to vary depending on the format which in turn depends upon the nature of the operand - for example a register identifier may be very much smaller than an instruction displacement.

In Figure 1 a first source code module 1 written in assembly language is input to assembler 2 to provide an object code module 3 which is in machine language and is directly analogous to the assembly language source code module. A second source code module 11, in assembly language is input to an assembler 12 to provide an object code module 13. It will of course be clear to those skilled in the art that more than two source code modules may be provided and that the same assembler could be used for each of a plurality of source code modules, the modules being assembled sequentially.

The source code modules provide an input to a linker 4 which may also receive an input from an object code library 6. The function of the linker includes binding those operands which are external symbols to addresses so that the object code modules cooperate together to form

executable machine code. Some code modules from the library will be required to effect this, in the case that the symbolic references are to objects in the library. The linker thus performs the function of a link editor.

In the prior art, as mentioned above, the assembler which is typically a microprocessor program run on a host microprocessor, requires hard-coded information to enable it to transform an assembly language instruction into its machine code equivalent. For example, if the assembly language instruction has the effect of writing information to a register, the corresponding machine code instruction may only allocate a relatively small number of bits to the bit field which the microprocessor will use to determine which register, whereas if the instruction is to write to memory, a much larger bit field to permit the memory address to be encoded may be provided. Thus, in the prior art the programmer of the assembler might have to specify both the size and location of each bit field in each instruction, indexed by, for example, the opcode as hard-coded information into the assembler. Although this may be acceptable where the instruction set architecture is fixed and constant, it can cause severe difficulties when a microprocessor or series of microprocessors is under development.

Referring to Figures 2-6, an embodiment of the invention will be described in which hard-coding of the assembler is reduced, thus allowing an assembler to track changes in the instruction set more readily. In a preferred embodiment the assembler is able to automatically track changes in the instruction set as they occur.

The arrangement shown in Figure 2 has some similarities to that of Figure 1 in that it shows a source code module 10 which is applied to an assembler 20 which outputs object code to a linker 40, which receives inputs of other object

code modules, shared libraries and the like to provide an executable program 50.

The assembler 20 consists of a translation device 21 which responds directly to the source code 10 to provide an encoded output 22 which represents a direct transliteration of the source code. The source code is also input to a fetch unit 23 whose function is to address a descriptor file 24 with information derived from the source code currently being translated to provide output information 25 representative of constraints due to the instruction set architecture. The output information 25 is applied to a control device 26 which operates on the translated information 22 to constrain that information to conform with the requirements of the instruction set. The output information 25 is also applied to a data transfer device 27 whose output is combined with the constrained translated information to provide an assembler output 28.

Generally speaking, the data conforming to the source code is provided from the control device 26 whereas the data transfer device 27 provides the information directly to the linker 40. This enables the linker to perform operations on the data, again determined by the instruction set. If for example the instruction requires scaling to be effected, this can be achieved by providing the relevant scaling factor via the data transfer device to the linker. This happens for example when the scaling is to be applied to an external symbol - i.e. a symbol whose value is known only at link time.

In this embodiment, the descriptor file is derived manually from manipulation and inspection of the instruction set 30 but in a preferred embodiment, a utility program 31 is used to access instruction set architecture data 30 to provide the descriptor file 24.

Figures 3 and 5 show two exemplary instructions, each of which are in fact 32 bits long, although fewer than this number of bits is shown. In Figure 3 the instruction includes an opcode A which requires two source registers RS1 and RS2 as well as a destination register (not shown). In this instruction, the first source register has a starting bit position of zero and a finishing bit position of 5 and the second source register has a starting bit position of 14 and a finishing bit position of 19. The bits between bits 5 and 14 are either empty or serve a function such as indicating an instruction format.

Referring to Figure 5 a second instruction has an opcode of B and in this case requires immediate data IMM1 starting at bit position 0 and running to bit position 7, and a destination register whose numeric identity is specified by bits 14-18.

It will be seen from this example that a substantial part of the output of the assembler requires alignment with the instruction set both from the point of view of the starting position of a particular bit field and the number of bits available for that bit field.

The instruction set database comprises information about the bit fields of each instruction. For instance it may be decided to scale (divide) by two on an address operand in order to use a smaller bit field. This information (in general the encoding function) is used by the assembler and is passed to the linker for use with external symbols.

By further providing the decoding function (e.g. multiply by two, in the example above,) a user's source code can be checked for the error of specifying an unencodeable value, such as an odd address in this example, or a register whose number is out of range.

The present invention provides information descriptive of the instruction set in the descriptor file 24 shown in Figure 2 a part of which is diagrammatically shown in Figures 4 and 6. In each of these Figures, the top line represents an identifier for the operand of concern, in Figure 4 "1A" represents RS1 and "2A" represents RS2 whereas in Figure 6 "IB" represents IMM1 and "DB" represents RD1, each of these being associated with the relevant bit field. Line 2 in each of the Figures shows the starting bit position for the associated operand, and the third line shows the bit length of the operand. The fourth line shows the end bit of each operand.

For each instruction the syntax, eg the mnemonic and its equivalent is derived and stored and for each bit field in the instruction the type of information of each bit field is derived and stored. For instance "12" for a particular opcode, all φ's for a reserved field, values 0-15 for a register, address/4 for a symbolic value.

The assembler then accesses the stored information to enable it to accept assembly language instructions, from for example source code modules, and encode those instructions to provide a translated output in machine language.

As an example, for a microprocessor having a set of 16 registers, it is decided to define an instruction for loading a value from external memory to a specified one of the set of registers. Such an instruction is given the mnemonic MOV and the particular register is defined as Rn, with n=0-15. It is further decided that bits 8-12 will contain the pattern 10110 for this instruction in a 2 byte instruction length. The address is to be encoded in bits 1-7 and the register number is specified in bits 13-16.

This data is stored as the descriptor file. When supplied with user source code of the form:

MOV fred, R12 ("fred" being an external symbol)

the assembler accesses the descriptor file to enable it to transliterate the assembly language into machine language, filling in the opcode and register. Moreover the necessary data are supplied from the descriptor file to enable the linker how to patch in "fred".